Floating Point Research Summary

Brian Crafton
crafton.b@husky.neu.edu

Research Start Date: 05/02/2016
Research End Date: 07/08/2016

Professor/Supervisor: Prof. Miriam Leeser

Introduction:

In theory floating point arithmetic is associative, meaning that reordering particular operations should have no effect on the output. However this is not the case when computing such operations on a computer. Compilers handle code differently and may order instructions differently across platforms and reordering of operations can cause different results. Another source of differences can be due to the use of a fused multiply add instruction which use a different number of bits between the multiplication and addition.

In this research opportunity it was my job to run different OpenCL benchmarks across different hardware platforms and collect data. Initially there was 6 platforms that were considered for this research. The first 5 of these belong to the SHOC benchmark suite. The last benchmark, SOR, was taken from the SciMark 2.0 benchmarks. The 6 benchmarks are listed below along with a description taken from their respective wikis (see appendices).

These 5 benchmarks are listed below along with a description taken from the shoc wiki.

1. MD: Measures performance for a simple nbody pairwise computation (the Lennard-Jones potential from molecular dynamics)
2. SPMV: Measures performance for sparse matrix-dense vector multiplication
3. Reduction: Measures performance for a large sum reduction operation using single precision floating point data
4. Scan: Measures performance for an exclusive parallel prefix sum of single precision floating point data
5. Stencil2D: Measures performance for a 2D, 9-point single and double precision stencil computation (includes PCIe transfer)
6. SOR: Jacobi Successive Over-relaxation on a 100x100 grid exercises typical access patterns in finite difference applications.

Of the 5 benchmarks from shoc, only 3 have been used as the results of the other 2 do not produce output data that is ideal for the tests we are running. Scan and Reduction both do not make use of floating point variables. The other three benchmarks make heavy use of floating point arithmetic and all of them show differences across the hardware platforms. The last benchmark, SOR, needed to be converted to OpenCL because it was written in C originally. Because the kernel was called on each index just like matrix multiplication the same host code used for matrix multiplication could be used for SOR. The only difference was the kernel needed to change.

In each of these benchmarks floating point operations were done on large sets of data and resulted in a matrix or vector. Because these benchmarks were implemented in OpenCL it was necessary to send and receive data from the host and device machines. The data that was sent was whatever was needed by the device to compute its kernel. The result of this kernel was transferred back for verification or use in another computation. This was the data that was collected and dumped to a csv file because this was the data that was computed on the device (hardware platform) and can be compared to other devices (hardware platforms).

The data that was collected were large sets of data, and hard to visualize or analyze. In order to facilitate this problem a script was created to analyze the data easier. The script creates 4 different forms of looking at the data.

1. Creates a CSV file of each comparison of hardware platforms.
2. Creates a histogram binning each cell in the different matrix or vector
3. Creates a table showing whether the output matrices for a hardware comparison are identical.
4. Displays the indexes largest differences (cell values) of each hardware comparison and displays the values of the same indexes off the other hardware comparisons.

Results/Analysis:

The experiments conducted for this research consisted of running floating point intensive applications in OpenCL code across different architectures to explore how results can vary across different hardware and compilers. For this section I will look mostly at SOR, but the same analysis and data are available for the other 3 benchmarks. The four benchmarks that were chosen to be used for this experiment are displayed below

Table 1: Boolean result difference matrix (1 = different, 0 = same)

	md	sor	spmv	stencil2d
('amdcpu', 'amdgpu')	1	1	0	1
('amdcpu', 'intel')	0	1	0	0
('amdcpu', 'nvidia')	1	1	1	1
('amdgpu', 'intel')	1	1	0	1
('amdgpu', 'nvidia')	1	1	1	1
('intel', 'nvidia')	1	1	1	1

From this matrix it is clear that there are quite a few differences across the hardware platforms. Each hardware platform had different results for SOR. Some of the differences are huge (relatively) and can be seen in the table below.

Table 2: Largest (absolute) differences for AMD CPU and AMD GPU (SOR)

	[31, 20]	[63, 96]	[64, 9]	[63, 26]	[95, 41]
('amdcpu', 'amdgpu')	-0.257004	0.263617	-0.269228	0.272479	0.273405

('amdcpu', 'intel')	-0.256658	0	0	0.272479	0.273404
('amdcpu', 'nvidia')	-0.253581	0.263617	0	0.000002	0.273404
('amdgpu', 'intel')	0.000346	-0.263617	0.269228	0	-0.000001
('amdgpu', 'nvidia')	0.003423	0	0.269228	-0.272477	-0.000001
('intel', 'nvidia')	0.003077	0.263617	0	-0.272477	0

Figure 2 displays the indexes of the 5 largest largest differences for the AMD GPU / AMD GPU pair and then shows the differences in these indexes for the rest of the hardware pairs. In some cases the result is 0 for other pairs, and sometimes it is of similar magnitude. This graph only shows the indexes of largest differences for AMD CPU / AMD GPU, but the indexes of the largest differences for the rest of the pairs are available in the git repository. It is interesting to note that all the largest differences fall somewhere between (0.25, 0.28) or (-0.28, -0.25).

Figure 2 showed the absolute differences between the pairs. Another interesting way to look at the data is the relative difference. By choosing a “control” hardware, it is possible to look at the percent difference for each hardware versus the control hardware. Table 3 shows the 5 largest differences between Intel and AMD CPU and the values in the same indexes for other hardwares versus Intel.

Table 3: Largest (relative) differences for AMD CPU and AMD GPU (SOR)

	[95, 41]	[31, 25]	[95, 8]	[31, 4]	[31, 95]
amdcpu	1.0128812605	1.0801932457	1.1127143594	1.3306989639	1.4798436178
amdgpu	3.70470534616E-06	4.65428007596E-06	1.1127143594	1.3306989639	0
nvidia	0	0.0002187512	0	0	0

Relative error is calculated by:
(Experimental error - Control Error) / Control Error

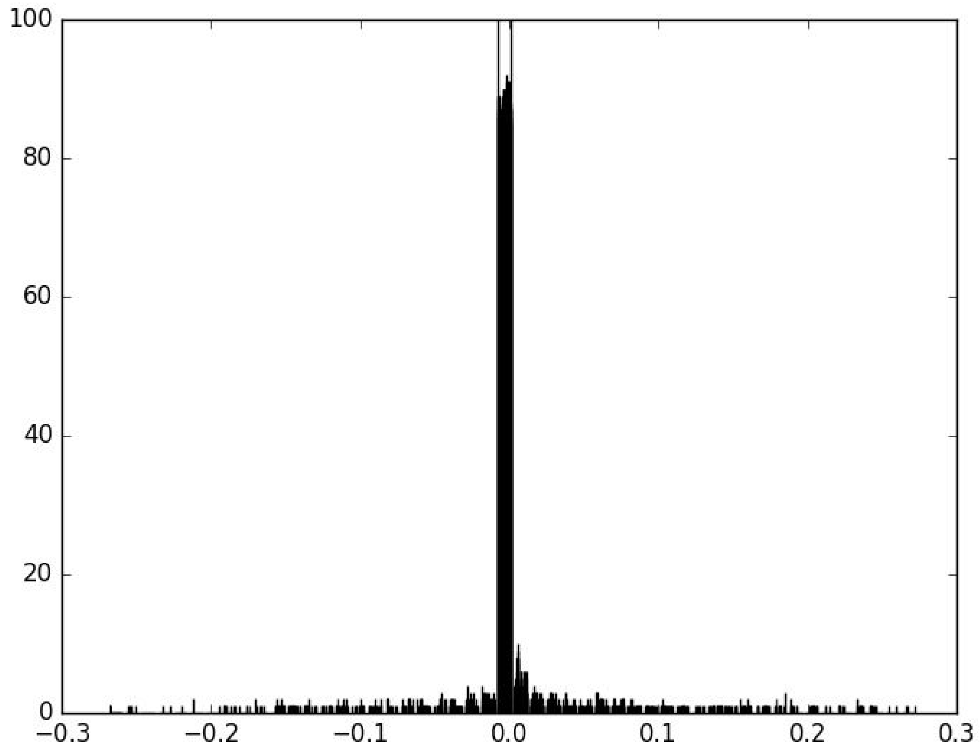
Figure 3 shows that in one case a value was off by 150%. The same indexes for the other hardwares do not have this huge error, but they have large errors of their own versus Intel in their 5 largest differences available in the git repository.

The data collected from the benchmarks can be visualized in two other different ways.

1. Looking at the difference of the two output matrices
2. Looking at the histogram of all values

I will not include the whole csv difference matrix in this, but it is available in the git repository. The histogram simply takes the output values and uses 50 bins and creates a graph of these values. An example of a histogram is the graph for the AMD CPU / AMD GPU different matrix. In the graph below the 10,000 cells from the 100x100 difference matrix are binned. It is clear that most of these difference are roughly 0 or are pretty close to it, but there are quite a few outliers that go
all they way up to +- 0.28.

Figure 1: AMD GPU / AMD GPU histogram



There are so many differences between hardware platforms running SOR. The reason for this is different orderings of instructions by the compiler. For this reason it is important to look at the assembly and interpret some of the differences. In OpenCL there are different methods for dumping this assembly code and it was done for the NVidia and AMD architectures, but the Intel one requires a special compiler that I did not have access to. Looking at the assembly code, it is clear that the assembly codes are definitely different, and because floating point arithmetic is not associate it is likely that there will be differences. Another thing that can be noted, is that the two GPU architectures use a fused multiply add (MAD) instruction while the CPU architecture does not.

Appendices / Links:

1. Benchmarks
    1. The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite
       https://github.com/vetter/shoc/wiki
    2. SciMark 2.0
       http://math.nist.gov/scimark2/
2. Github repository
   Link: https://github.com/bcrafton/FloatingPointResearch.

This repository contains all the data, code, and scripts as well as the tables, histograms, and matrices generated from the data and scripts.

1. The script **research.py** looks at relative directories for the data resulting from the code and automatically generates the tables, histograms, and matrices for analysis and visualization.
2. The **data** directory contains the input and output data for the benchmarks that were run. The input data is the data that is sent to the kernel, and the output data is the data that is returned from the kernel.
3. The **analysis** directory is the directory containing the tables, histograms, and matrices created by the script.
4. The **assembly** directory contains the assembly code for the SOR benchmark on AMD and NVidia hardware platforms.
5. The **shoc_src** directory contains the modified source code from each of the benchmarks that dump the input and output data.

3. Worklog

Link:
https://docs.google.com/document/d/1kXrv0-rE4m5RZoxib7EVf-fY6slhVhYrA77RbXGI5vk/edit?usp=sharing

This doc contains daily notes I made when configuring the benchmarks, writing the script or modifying the benchmarks. There were quite a few issues with working with the clusters, trying to run on one of the hardware platforms, and configuring benchmarks. The answer to a future problem might be findable in the worklog or just by asking me.

4. Additional Notes

The section will just provide thoughts or things were mentioning.

1. Of the 4 benchmarks the only 2 that were verified to all have the same input were SOR and SPMV. For both of these benchmarks I understood well what was going on in the code. The others I did not heavily explore but rather looked for where the data was being pulled from the device and then dumping it to the output csv.
2. In order to add any additional benchmark data, one only needs to follow the naming format and place the input and output data in the input and output data folders in the git repository.
   Naming format <*hardware-platform*>_<*benchmark-name*>
3. Although SPMV is shown as one benchmark there are 3 different implementations of SPMV available in the SHOC benchmark suite. This would provide more sample data showing the floating point differences across hardware.
4.

A decent amount of time I spent on this research opportunity was spent:

1. Figuring out how to run OpenCL on one of the 4 hardware platforms
2. Figuring out how configure the benchmarks on the clusters.
3. Figuring out how to analyze data or what data to analyze
4. Writing a script to generate visualization tools

Now that there is a framework and standard in place for collecting and analyzing the data, more time can be spent looking for new benchmarks to provide more samples for analysis.

The Current procedure for collecting useful data.

1. Modify benchmark to dump input and output data to csv
2. Setup, compile, and run benchmark on each of the hardware platforms
3. Verify that all input data is the same
4. Correctly name input and output csv files
5. Run the script